# Language and Tools

.Net Framework 4.5

C#

LINQ

Visual Studio 2013, Professional Edition

SQL Server 2014

Microsoft Word (Documentation Purpose)

# Concepts tried to Demonstrate

S.O.L.I.D principles of Object Oriented Programming

Test Driven Development (Used in developing Queue Puzzle)

Unit Testing (Used in developing Queue Puzzle)

C#

T-SQL

# IQueue Interface

A queue of generic type is implemented with very basic functions in a queue.

## The Interface

Description

Following properties and methods are defined in the interface

|  |  |  |
| --- | --- | --- |
| **Name** | | **Description** |
| Properties | |  |
|  | Length | Readonly property which is used to return the number of items currently in the queue |
|  | IsEmpty | Readonly property which checks if the queue is currently empty or not |
|  |  |  |
| Methods | |  |
|  | EnQueue | Inserts an item at the top of the queue |
|  | Clear | Empties the queue |
|  | Dequeue | Returns the first element that was inserted in the queue and removes this item from the queue |
|  | Peek | Returns the first element that was inserted in the queue |
|  | Reverse | Reverses the current queue |
|  | ReverseCopy | Returns the reverse copy of the current queue but keeps the current queue intact. |
|  | Contains | Checks if an element exists in the queue or not |

## CustomQueue Class

This class implements the IQueue Interface. Queue is implemented through the list of Generic. Since it uses System.Linq namespace, implementation is easy as many linq functions are available.

## CustomQueueWithArray

This class implements the IQueue Interface but implements queue through array and does not use the System.LINQ namespace so that the built in Linq functions are not available and the requirements have to be explicitly coded.

## Tests

Unit Tests has been written for every function of the interface and it has been ensured that the unittests pass successfully.

## Exception Handling

Since the CustomQueue is a generic class, assumption is made that the errors be handled at the implementation level. Minimal exception handling has been implemented.

# Menu Puzzle

## Objective

Produce a collection of root items with each item holding a collection of child items.

## Implementation

1. A menu class is created for the purpose of simulating the database
2. A custom class has been created to represent the tree View. It consists of List of Menu objects (instantiation of class created in step 1 above) representing the child elements of the current node.
3. A repository of menu items has been created using the menu class

## Dummy Menu Used

Root

Edit

File

New

Open

Project

Website

File

Design

Flowchart

DFD

Cut

Copy

Paste

Paste Special

JSON as Classes

XML as Classes

## Algorithm

Step 1:

Create a root node of the tree View Structure

Step 2:

Loop through the menu table (in my case the menu repository implemented through list) such that the ParentId is 0 or null (root element do not have parents associated with them)

1. Create a tree View Node for each root menu item
2. Call Populate Tree function to populate all the child elements (branches and leaves) for this element

Step 3: Populate Tree Function

If the current Tree View is a leaf (i.e. branches with no child elements/final menu elements) return

Else push the current tree node in stack (implemented automatically by .NET framework when using the recursive function call) and call the Populate Tree function to find the children of the current node

Step 4: Print the tree View

## Error Handling

Since the errors that might occurred could not be fixed or recovered in the runtime, they are simply logged to the log file for review by the programmer or concerned authority to be fixed later.

IExceptionLogger interface has been implemented to log the errors. Logging of errors is done by ExceptionLogger Class which implements IExceptionLogger. This has been done so in order to separate the concerns as reading and writing file is not the task of the main program.

Note: Dependency inversion has not been properly implemented as it would require dependency injection which would make the code complex.

Error in the ExceptionLogger itself is swallowed back as there can be no other approach.

## Alternatives

There can be various alternatives like implementing the tree with the linkedlist, but I preferred to construct a custom tree structure of my own.

## Testing

Unit Tests have not been written as the program is already using a dummy repository and unit tests would have made it redundant.

# Puzzle 3

## Part A

select Convert(varchar(20),OrderDate,106) ORDERDATE,SUM(Quantity)

From order a

inner join orderDetail b on a.OrderID=b.OrderID

GROUP BY Convert(varchar(20),OrderDate,106)

HAVING SUM(QUANTITY)>25

## Part B

select Convert(varchar(20),OrderDate,106) as Date,Count(\*) as "No of Blank Orders"

FROM order a

left join orderDetail b

on a.OrderID=b.OrderId

WHERE b.OrderId IS NULL

GROUP BY Convert(varchar(20),OrderDate,106)

HAVING SUM(COALESCE(Quantity,0))=0

# Disclaimer

All the codes have been written by developer himself and no reference has been taken from the internet/books or anywhere, it purely is based on the experience of the developer.